# Scalable Implementation of Dependence Clustering in Apache Spark

Elena Ivannikova
Department of Mathematical Information Technology
University of Jyväskylä
POBox 35 (Agora), 40014 Jyväskylä
Email: elena.v.ivannikova@student.jyu.fi

***Abstract*—This article proposes a scalable version of the Dependence Clustering algorithm which belongs to the class of spectral clustering methods. The method is implemented in Apache Spark using GraphX API primitives. Moreover, a fast approximate diffusion procedure that enables algorithms of spectral clustering type in Spark environment is introduced. In addition, the proposed algorithm is benchmarked against Spectral clustering. Results of applying the method to real-life data allow concluding that the implementation scales well, yet demonstrating good performance for densely connected graphs.**

## I. Introduction

Efficient analysis and processing of large-scale datasets requires scalable algorithms and computational frameworks. To address this challenge, distributed large-scale data processing frameworks have emerged in recent years. One of the most widely-used frameworks/platforms for processing large-scale data, Hadoop [1] is an open source implementation of MapReduce [2]. Despite performing relatively well for offline data, it handles real-time stream data poorly. Moreover, Hadoop normally processes data from the disk which is inefficient for data mining applications that often require numerous iterations. It has been reported by a number of works that Hadoop-run algorithms sustain significant performance loss due to disk I/O operations and network communications [3], [4].

Apache Spark [5] is a more recent open-source distributed framework for data analytics which enables, among other things, fast and efficient processing of large streams of data. The key features of Spark are in-memory computations and fault-tolerance. Spark adopts Resilient Distributed Dataset (RDD) [5], a distributed memory abstraction which supports two types of operations: transformations and actions. Transformations define a new RDD based on the existing one, and actions either return a value to the driver program or export data to a persistent storage. When a transformation is executed a new RDD is created with its records distributed across the main memory. An action operation causes each node to process its local set of records and return the result. Spark also supports in-memory caching of datasets which prevents slow disk reads and performs much faster compared to Hadoop-like systems.

Spark proved itself to be effective for performing machine learning and data mining tasks involving big datasets. Recently, many works refer to Spark as a tool for creating competent solutions for data analysis. Many clustering and anomaly detection algorithms have been developed or adapted to Spark due to its efficience and high performance. Apache Spark MLlib library [6] has a number of implemented clustering algorithms such as $k$-Means, bisecting $k$-Means, Gaussian mixtures (GMM), and Power Iteration Clustering (PIC). Some other famous clustering algorithms have been implemented in Spark framework but they do not belong to core Spark libraries, e.g. CURE [7] and a scalable random sampling variation of fuzzy $c$-Means [8].

Spectral clustering methods [9] detect structure of data distribution based on information aquired from the spectrum of the data affinity matrix. One of the hardest computational tasks usually seen in spectral clustering methods is the necessity to solve an eigenvalue problem of the Laplacian matrix derived from the data affinity matrix. Therefore, spectral clustering algorithms often need to be adapted when applied to large-scale datasets. One way to improve efficience is reducing size of the affinity matrix. Another solution is avoiding recomputation as new data points arrive, making the method suitable for real-time algorithms [10], [11]. A number of methods have been developed in order to avoid high complexity caused by calculating spectrum of the Laplacian matrix and to make spectral clustering applicable for large scale datasets. Among them are approximation methods which perform spectral clustering on either a subset of representative data points or randomly sampled values of the affinity matrix and extend the obtained results to the remaining data points [12], [13], [14], [15].

PIC [16] is an efficient method for clustering data which embeds data points in a low-dimensional subspace derived from the affinity matrix, similarly to other spectral clustering methods. Under the hood, PIC applies 1D $k$-Means to a non-converged top ranked eigenvector. In PIC embedding results in approximation of a linear combination of all eigenvectors of a normalized affinity matrix where eigenvalues serve as coefficients of the linear combination. Such approach is efficient compared to traditional spectral clustering due to performing only a small number of matrix-vector multiplications instead of running iterations until convergence.

Dependence Clustering (DEP) [17] is a method which considers geometric structures of data and is based on maximizing the *group dependence* measure. The method assumes that any two nodes in the graph can be connected through Markovian

transitions that enables calculation of *dependence distance* [18] between graph nodes in a certain evolution step. The level of connectivity scale in group assignment can be adjusted improving the flexibility in regulating the level of detail. This approach determines the optimal number of clusters while dividing the data into clusters. This is particularly important for exploratory research related to real world applications with an unknown number of clusters beforehand. In this paper, a Spark-based implementation of DEP [1] which allows better performance for analysis of big datasets is introduced.

The main contributions of the paper consist of

1) Introducing and implementing in Apache Spark a fast approximate diffusion that enables spectral clustering type algorithms in Spark environment.
2) Implementing a scalable version of DEP in Apache Spark framework.

The rest of the paper is organized as follows. Section II provides descriptions of the main concepts and methods used in the paper. Sections II-A - II-C explain the proposed DEP clustering algorithm. Meanwhile, Spark implementation is described in Section II-D. Section III is devoted to the experimental results. It introduces evaluation metrics, data sets and results of the performance tests. Finally, Section IV completes the paper with conclusions and discussions.

## II. Methods

### A. Preliminary concepts

We start with a graph defined by an affinity matrix A [19]. The graph is formed by a set of data points $\Omega = \{x_i | i = 1, ..., N; x_i \in \mathbb{R}^n\}$. Therefore, the matrix A with entries from $\mathbb{R}$ representing pairwise similarities has size $N \times N$. We define a Markov chain on this graph by transforming A to the transition matrix P and the corresponding $t$-step transition matrix $\mathrm{P}^t : P^t_{i,j} = Pr(X_t = j | X_0 = i)$, where a probability variable $X_0$ represents the initial state and $X_t$ is a random walk representing a node at the $t$-th transition. The transformation from A to P is done by scaling rows of A so that elements in each row sum up to one. Assuming that the whole chain is ergodic and all transitions follow the Markovian property we define statistical dependence $D_{i,j,t} = Dep(X_0 = i, X_t = j)$ [18] by the following equation:

$$D_{i,j,t} = \frac{Pr(X_0 = i, X_t = j)}{Pr(X_0 = i) Pr(X_t = j)}. \quad (1)$$

Statistical dependence captures inter-dependency of the node in the initial state and the node at $t$-th transition. Let us denote a group assignment vector by $\mathbf{s} = [s_1, ..., s_N]$, where decision variable $s_i = 1$ if data point $i$ belongs to group 1 and $s_i = -1$ if it belongs to group 2. Note that in such notation $(s_i s_j + 1)/2$ is 1 if $i$ and $j$ are in the same group and is 0, otherwise. Thus, given $\mathbf{s}$ and $t$, the group dependence for a particular choice of $\mathbf{s}$ is defined as follows:

$$D_t = \frac{1}{2} \sum_{x_i, x_j \in \Omega} \big(D_{i,j,t} - d_0\big)(s_i s_j + 1), \quad (2)$$

[1] https://github.com/Korelena/spark

where $D_{i,j,t}$ is defined in (1), $d_0 = 1 + \epsilon_d$ is the baseline dependence level which is usually set to 1 and $\epsilon_d$ is a dependence margin parameter. A constant $d_0$ effectively normalizes the statistical dependence for each pair of points equaling zero when points are considered to be independent. More details about parameter settings and optimization procedure can be found in [17].

Group dependence described above captures aggregate inter-dependency of points within the groups along with considering the geometrical structure of the data. Hence, statistical dependence serves as a measure of closeness between data points.

### B. DEP for two groups

We start with describing the DEP algorithm for a simple case of bisecting a graph. An optimal clustering solution can be achieved through maximizing the group dependence measure $D_t$ by varying the group assignment $\mathbf{s}$ of all $N$ points. We constrain the norm of $\mathbf{s}$ to be equal to one: $||\mathbf{s}||_2 = 1$ assuming for simplicity that $\epsilon_d = 0$. Moreover, we relax the original formulation (2) so that elements of $\mathbf{s}$ become real as the actual optimization is carried out in the domain of real numbers $\mathbb{R}$. Then, we obtain a good partition by solving the following maximization problem:

$$\arg\max_{||\mathbf{s}||=1} D_t = \arg\max_{||\mathbf{s}||=1} \frac{1}{2} \sum_{i,j} (D_{i,j,t} - 1)(s_i s_j + 1).$$

It can be shown [17] that the problem above is equivalent to

$$\arg\max_{||\mathbf{s}||=1} \mathbf{s}^T (\mathrm{P}^t (\mathrm{B}^{(t)})^{-1} - \mathbf{1}\mathbf{1}^T)\mathbf{s},$$

where diagonal matrix $\mathrm{B}^{(t)} : B^{(t)}_{j,j} = Pr(X_t = j) = [x_0^T \mathrm{P}^t]_j$, $x_0$ is the initial probability vector representing prior information related to the initial states, and $\mathbf{1}$ is the all-ones vector. This constrained optimization problem can be solved by using one of the standard eigendecomposition numerical algorithms, e.g. Power iteration or Arnoldi iteration [20], [27].

Finally, division of the data points is made based on the signs of the eigenvector corresponding to the largest positive eigenvalue of G defined by

$$\mathrm{G} = \mathrm{P}^t (\mathrm{B}^{(t)})^{-1} - \mathbf{1}\mathbf{1}^T. \quad (3)$$

The nonexistence of positive eigenvalues means no possible benefits for increasing $D_t$ from further divisions.

### C. DEP for multiple groups

To obtain divisions to multiple groups, we apply a standard subsequent division approach [21]. At every step we consider binary divisions of every group already found during the previous iterations, following the DEP algorithm described in Section II-B. Among possible divisions we proceed with the one that results in the maximal increase of group dependence for the whole dataset, effectively performing greedy search.

For the purpose of preventing the algorithm from defining too small or unclear clusters, we introduce a dependence gain parameter $\delta_d \in [0, 1]$. The minimal dependence gain required to split a cluster into two subclusters is calculated as $\Delta_d =$

$\delta_d \sum_{g_{i,j}>0} \mathrm{G}$, where G is defined in (3). Let us denote a set of points that belong to a split candidate cluster by $\Omega_C \subseteq \Omega$. We denote the within-cluster group configurations before and after the division of $\Omega_C$ into two subclusters by $\mathbf{s_C}$ and $\mathbf{s'_C}$, respectively. The division into the subclusters proceeds if $D_t(\mathbf{s'_C}) - D_t(\mathbf{s_C}) > N\Delta_d$, where $D_t$ is defined in (2) and $N$ is size of $\Omega$. Note that all elements from $s_C$ are equal to one.

### *D. Spark implementation*

Our implementation is written in Scala and is inspired by the implementation of PIC [16]. In addition, we implemented a Python wrapper. We used GraphX Spark API as a backend to store sparse dependence matrices in a distributed manner and perform computations [22], [23]. GraphX exposes data-parallel and graph-parallel paradigms that allow a versatile set of operations to be done within a single framework.

Consider the data is represented as a graph $G =< V, E >$ where $V$ and $E$ define nodes and weighted edges, correspondingly. We assume that data chunks of order $O(|V|)$ can be stored on a single machine. We also assume that sparsity of data allows redistributing the data across $O(log(|E|)$ machines. Typical real world large scale graphs tend to respect skewed power-law distributions of node degrees [24], [25], [26].

For computing the largest eigenvector of a matrix we used the Power Iteration (PI) method [27]. PI is an iterative method that works as follows. Starting with an arbitrary initial vector $\mathbf{v}_0 \neq 0$ it performs an update $\mathbf{v}_{k+1} = c\mathrm{A}\mathbf{v}_k$, where A is the affinity matrix, $c = \|\mathrm{A}\mathbf{v}_k\|^{-1}$ is a normalizing constant. Due to simplicity of its operations, as only matrix-vector multiplications are performed, the PI method can be used as an integral part of scalable large-scale data analytics solutions.

The proposed implementation of the DEP algorithm is summarized in a pseudo-code below.

**Require:** Normalized graph $G =< V, E >$ cf. (3), $\Delta_d$
1: Initialize a list of data structures holding information about each group $L = [l_1]$, where the first group includes all nodes $l_1.V = V$ and has group dependence $l_1.D_t = \sum_{e_{i,j} \in E} e_{i,j}$.
2: **while true do**
3: maxDepGain = 0
4: maxGroup = -1
5: **for all** $l_i$ in L **do**
6: Take a subgraph $G' =< l_i.V, E' >\subseteq G$
7: Apply PI algorithm to $G'$ and obtain the highest ranked eigenvector s.
8: Split nodes of $G'$ into two sets: $V'_1$ for s $< 0$ and $V'_2$ for s $> 0$
9: Take two subgraphs of $G'$: $G'_1 =< V'_1, E'_1 >$ and $G'_2 =< V'_2, E'_2 >$
10: Set $D_t^1 = \sum_{e_{i,j} \in E'_1} e_{i,j}$ and $D_t^2 = \sum_{e_{i,j} \in E'_2} e_{i,j}$
11: Compute a dependence gain of this split as depGain = $D_t^1 + D_t^2 - l_i.D_t$
12: **if** depGain $>$ maxDepGain **then**
13: maxDepGain = depGain
14: maxGroup = $i$
15: **end if**
16: **end for**
17: **if** maxDepGain $> N\Delta_d$ **then**
18: Initialize entries for the two new subgroups of maxGroup in L
19: **else**
20: **return** L which contains entries for all found groups
21: **end if**
22: **end while**

Another computational problem that we addressed was a diffusion operator [28]. Iterating Markov transition matrix by taking powers of P has an effect of diffusing probability mass from the high potential regions to the potential lower ones. Since a direct multiplication of the large-scale sparse matrices is computationally demanding we approximate the effect of probability diffusion by the locally guided diffusion of affinity in the original affinity space. Namely, we design a procedure of joining disconnected nodes with the high transitive similarities, i.e. on the path $v_i \xrightarrow{e_{i,k}} v_k \xrightarrow{e_{k,j}} v_j$ between disconnected nodes $v_i$ and $v_j$ all the weights of the edges $e_{i,k}$, $e_{k,j}$ and the influx of the node $v_k$ are relatively high.

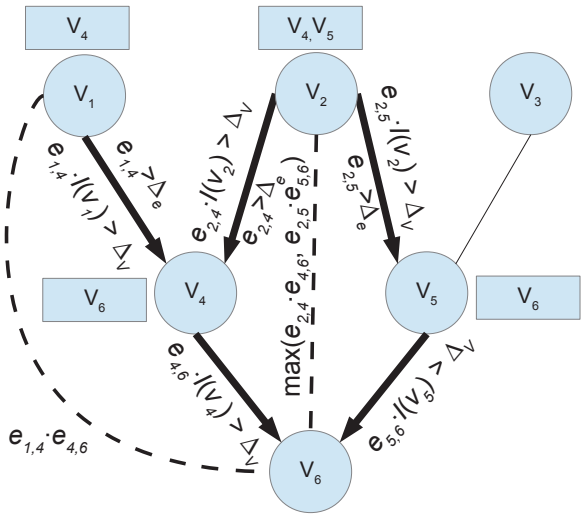


Fig. 1. Illustration of the diffusion procedure. Bold edges denote paths that satisfy constraints for a new edge addition. Expressions along the bold lines are the conditions that enabled emergence of new edges. Added edges are marked dashed. Expressions along the dashed lines determine weights of the newly formed edges. Node names in the boxes denote nodes that are aggregated at a particular node aside.

The procedure goes as follows (see Fig. 1). First, for every node $v_i$ we calculate an influx $I(v_i)$ as a sum of weights of all in-bound edges. Then at every node $v_i$ we aggregate a hash map of nodes $v_j$ such that there exists an outgoing edge from $v_i$ to $v_j$. IDs of the nodes $v_j$ serve as keys of the hash map and weights of the edges $e_{i,j}$ serve as its values. Entries of only those nodes are aggregated that pass a strength test $e_{i,j} \times I(v_i) > \Delta_v$. At the next step triples of edge, source and destination nodes are considered, i.e. hash maps of two nodes $v_i$, $v_j$ and their connecting edge $e_{i,j}$ are brought together. A

new edge $e_{i,k}$ is added to the graph if $e_{i,j} > \Delta_e$ and there is an entry for the node $v_k$ in the hash map of $v_j$. The weight of a newly formed edge becomes the maximum aggregation among all the paths that enable the new edge, i.e. $\max_{v_k}(e_{i,k} \times e_{k,j})$. This procedure is guided solely by the local neighbourhood information. Under the assumption of the graph sparsity and with a proper set of parameters it should scale well. Namely, $\Delta_v$ along with $\Delta_e$ control sizes of the hash maps at each node and the number of added edges. Thus, the procedure effectively simulates diffusion and, at the same time, the graph sparsity remains protected by allowing only strong edges to emerge.

## III. Experimental results

### A. Experimental environment

In our experiments we used two setups. The local setup is a standalone version of Spark that supports parallelization across multiple cores. The test server had the following characteristics: 1TB of RAM, 64 CPU cores (8 cores Intel Xeon e7-8837 2.67GHz per CPU). For our tests we only reserved 16 cores/executors with 4GB RAM each and 16 GB RAM for the driver program. The second setup is a cluster deployed on Amazon Web Services (AWS) that had 4 slave nodes with 4 cores each and 12.4GB RAM per slave reserved for Spark. A master node had 4 cores and 16GB RAM in total.

### B. Evaluation metrics

Given true clustering labels/categories, we used the following two evaluation metrics to measure performance of the algorithms.

**Purity** [29], [30] focuses on the frequency of the most common category in each cluster and is computed as follows. First, each cluster is assigned to the most frequent category in the cluster. Then the number of correctly assigned items is counted and divided by the total number of clustered items $N$. the Purity is defined by the following equation:

$$\mathrm{Purity}(\mathbb{C}, \mathbb{B}) = \frac{1}{N} \sum_k \max_j |c_k \cap b_j|,$$

where $\mathbb{C} = \{c_1, c_2, \ldots, c_K\}$ is the set of clusters and $\mathbb{B} = \{b_1, b_2, \ldots, b_J\}$ is the set of categories, $c_i$ is a set of labels assigned to the $i$-th discovered cluster, and $b_j$ is a set of $j$-th cluster categories.

**Inverse Purity** [30] focuses on the frequency of the most common cluster in each category and is defined by the following equation:

$$\mathrm{InversePurity}(\mathbb{C}, \mathbb{B}) = \frac{1}{N} \sum_j \max_k |c_k \cap b_j|.$$

Note, that Purity becomes higher when the number of clusters is large and reaches its maximum when each item gets its own cluster. Inverse Purity reaches its maximum when all items belong to a single cluster. Therefore, a combination of the two measures is normally used for more accurate results.

TABLE I
Diffusion test results reported for the Reuters-1856 dataset.

| Parameters | | | | Purity | Inverse Purity |
|---|---|---|---|---|---|
| p | t | $\Delta_e$ | $\Delta_v$ | | |
| 1 | 0 | - | - | 0.6775 | 0.8566 |
| 0.5 | 0 | - | - | 0.2947 | 1.0 |
| 0.5 | 2 | 0.3 | 85.0 | 0.6439 | 0.7766 |
| 0.5 | 1 | 0.3 | 85.0 | 0.2947 | 1.0 |
| 0.75 | 0 | - | - | 0.2947 | 1.0 |
| 0.75 | 1 | 0.3 | 70.0 | 0.6210 | 0.8336 |

### C. Reuters data set

In our tests we refer to the tag *Topics* of the Reuters dataset [31]. We prepared two datasets named Reuters-8852 and Reuters-1876. Reuters-8852 was formed in the following way. Among all topics the following ten were selected: *money-fx*, *grain*, *crude*, *interest*, *trade*, *ship*, *acq*, *earn*, *wheat* and *corn*. Initially, 9400 articles were taken where at least one of the selected topics was present. The affinity matrix for the articles was defined as a cosine similarity among $L_2$-normalized term frequency (TF) document vectors.

We cleaned the data in the following way. First, we extracted a subset of articles where only one of the selected topics was present. Next, we generated TF features following the procedure of removing punctuation, tokenization and stemming (Porter stemmer [32]). We skipped all the terms that appeared in more than 40% of documents. Among the obtained TF features we selected the 1000 most discriminative features according to the $\chi^2$ test [33], which effectively resulted in a reduced vocabulary of 1000 terms. Similarly, we generated TF matrix for each of the 9400 documents considering only terms from the reduced vocabulary. Finally, we obtained 8852 articles with at least one non-zero feature and built the affinity matrix A which we used for testing the algorithm.

Reuters-1876 was derived in a similar way except that we used 500 TF features and considered only four topics: *grain*, *crude*, *interest* and *trade*. This dataset initially contained 2000 documents before all the empty documents were removed. Finally, Reuters-1876 had 1876 documents.

### D. Diffusion test

To verify correctness of the diffusion procedure we run the tests on the Reuters-1876 dataset. Table I displays results of applying DEP to the Reuters-1876 dataset undergoing different subsampling and diffusion rates. Here the parameter $p$ stands for the sampling probability. The parameter $t$ is the number of diffusion iterations applied to a graph. Thus, $t = 0$ means no diffusion was applied. First, we run the DEP algorithm on the original Reuters-1876 dense graph data. Next, we run DEP on the same dataset which was subsampled first by a factor of two ($p = 0.5$) that significantly degraded clustering scores. Applying DEP to the subsampled by a factor of two data with diffusion under parameter settings: $t$=2, $\Delta_e$=0.3, $\Delta_v$=85.0 produced results comparable to the clustering results without

subsampling and diffusion. Moreover, running diffusion under the same parameter settings but only ones ($t = 1$) still resulted in degraded solution. The latter means that subsampling has made severe damage to the graph structure.

We also applied DEP with and without diffusion to a less degraded graph where only quater of the edges were randomly discarded ($p = 0.75$) to verify the effect of the diffusion scale parameter $t$. In this case running the algorithm without diffusion resulted in much less accurate results compared to the scores of a run with diffusion applied. In all the runs $\epsilon_d$=0.16, $\delta_d$=0.13. These results confirm that the diffusion procedure correctly reconstructs the graph structure.

### E. Scalability test

We verify scalability of our implementation using both local and cluster setups. To perform the test we first sparsify the Reuters-8852 dataset by dropping all the edges that have weights less than 0.35. After this step 3742377 edges and all 8852 nodes were retained. Next, we run the DEP algorithm for the successively reduced dataset measuring execution times (see Fig.2). We partitioned the dataset to 16 partitions. In this experiment, other parameters were set to the following values: $\epsilon_d$ = 0.35, $\delta_d$ = 0.00, $t$=0.

One can verify that the DEP algorithm scaled near linearly in terms of the number of computations with respect to the number of nodes. From Fig. 2 one can see that the difference in timings between local and cluster setups is nearly constant and is apparently caused by additional network communication in cluster setup.

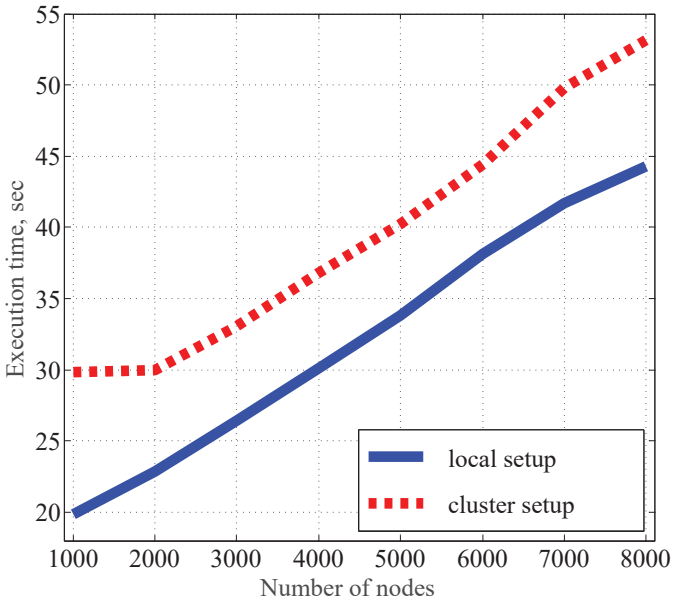


Fig. 2. Execution time of DEP as a function of number of nodes in the dataset.

### F. Performance comparison

We compared clustering results of the DEP and Spectral clustering [9] methods applied to the Reuters-1856 dataset.

TABLE II
PERFORMANCE COMPARISON RESULTS REPORTED FOR THE REUTERS-1856 DATASET. MEAN AND (STANDARD DEVIATION) ARE SHOWN FOR PURITY AND INVERSE PURITY.

| Method | Purity | Inverse Purity |
|---|---|---|
| DEP | **0.6775 (0.0)** | **0.8566 (0.0)** |
| Spectral clustering | 0.3070 (0.0079) | 0.6825 (0.2000) |

The results are displayed in Table II. For each method we reported mean and standard deviation values of Purity and Inverse Purity computed over 100 iterations. DEP was more accurate compared to Spectral clustering with regard to both measures. Moreover, low standard deviation implies that DEP was more stable compared to Spectral clustering.

## IV. CONCLUSION

In this paper we described a scalable Spark-based implementation of the DEP algorithm for clustering data points. The implementation is backed by the efficient Graphx API that supports graph-parallel and data-parallel paradigms. The method belongs to the class of spectral clustering algorithms and performs iteratively greedy binary splits to subgroups, thus, also resembling divisive hierarchical clustering scheme.

We introduced an approximate diffusion algorithm that acts over affinity data matrix and simulates Markov transitions. One should, however, carefully choose parameters in order to avoid memory overflow and to stay in bounded computational resources requirements. An interesting research direction would be to further explore various schemes for carrying out the diffusion.

The tests with real data show that the proposed implementation performs well. Moreover, our algorithm outperformed Spectral clustering which is a common, yet, strong benchmark in cluster analysis. The proposed algorithm can be applied for cluster analysis of large data sets. The potential applications of the algorithm span analysis of text, social networks and network security data, which is a focus of future research.

## ACKNOWLEDGMENT

This research was supported by the Nokia Foundation Scholarship funded by Nokia, Finland.

## REFERENCES

[1] "Apache hadoop," Available at http://hadoop.apache.org/.
[2] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," vol. 51, no. 1. New York, NY, USA: ACM, Jan. 2008, pp. 107–113.
[3] K. Kambatla and Y. Chen, "The truth about mapreduce performance on ssds," in *28th Large Installation System Administration Conference, LISA '14, Seattle, WA, USA, November 9-14, 2014.*, 2014, pp. 109–118.
[4] F. Pan, Y. Yue, J. Xiong, and D. Hao, "I/o characterization of big data workloads in data centers," in *Big Data Benchmarks, Performance Optimization, and Emerging Hardware: 4th and 5th Workshops, BPOE 2014, Salt Lake City, USA, March 1, 2014 and Hangzhou, China, September 5, 2014, Revised Selected Papers*, J. Zhan, R. Han, and C. Weng, Eds. Springer International Publishing, 2014, pp. 85–97.

[5] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 2–2.

[6] X. Meng, J. K. Bradley, B. Yavuz, E. R. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "Mllib: Machine learning in apache spark," *CoRR*, vol. abs/1505.06807, 2015.

[7] L. Haoxi, D. Min, T. Xue, B. Sheng, C. Dan, and Q. Rongcai, "The spark-based framework for mobile network data and cluster analysis on mobile users' behaviors," in *2015 IEEE International Conference on Robotics and Biomimetics, ROBIO 2015, Zhuhai, China, December 6-9, 2015*, 2015, pp. 487–492.

[8] N. Bharill, A. Tiwari, and A. Malviya, "Fuzzy based clustering algorithms to handle big data with implementation on apache spark," in *Second IEEE International Conference on Big Data Computing Service and Applications, BigDataService 2016, Oxford, United Kingdom, March 29 - April 1, 2016*, 2016, pp. 95–104.

[9] F. R. K. Chung, *Spectral Graph Theory*. American Mathematical Society, 1997.

[10] H. Ning, W. Xu, Y. Chi, Y. Gong, and T. S. Huang, "Incremental spectral clustering by efficiently updating the eigen-system," *Pattern Recognition*, vol. 43, no. 1, 2010.

[11] T. Kong, Y. Tian, and H. Shen, "A fast incremental spectral clustering for large data sets," in *12th International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2011, Gwangju, Korea, October 20-22, 2011*, 2011, pp. 1–5.

[12] D. Yan, L. Huang, and M. I. Jordan, "Fast approximate spectral clustering," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. New York, NY, USA: ACM, 2009, pp. 907–916.

[13] L. Wang, C. Leckie, K. Ramamohanarao, and J. C. Bezdek, "Approximate spectral clustering," in *Advances in Knowledge Discovery and Data Mining, 13th Pacific-Asia Conference, PAKDD 2009, Bangkok, Thailand, April 27-30, 2009, Proceedings*, 2009, pp. 134–146.

[14] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, "Spectral grouping using the nyström method," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 2, pp. 214–225, Jan. 2004.

[15] H. Shinnou and M. Sasaki, "Spectral clustering for a large data set by reducing the similarity matrix size," in *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2008, 26 May - 1 June 2008, Marrakech, Morocco*, 2008.

[16] F. Lin and W. W. Cohen, "Power iteration clustering," in *ICML*. Omnipress, 2010, pp. 655–662.

[17] H. Park and K. Lee, "Dependence clustering, a method revealing community structure with group dependence," *Know.-Based Syst.*, vol. 60, pp. 58–72, Apr. 2014.

[18] K. Lee, A. Gray, and H. Kim, "Dependence maps, a dimensionality reduction with dependence distance for high-dimensional data," *Data Min. Knowl. Discov.*, vol. 26, no. 3, pp. 512–532, May 2013.

[19] U. Ozertem, D. Erdogmus, and R. Jenssen, "Mean shift spectral clustering," *Pattern Recognition*, vol. 41, no. 6, pp. 1924 – 1938, 2008.

[20] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical Mathematics*, ser. Texts in Applied Mathematics. Paris, FR: Springer, 2007.

[21] M. E. J. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[22] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A resilient distributed graph system on spark," in *First International Workshop on Graph Data Management Experiences and Systems*, ser. GRADES '13. New York, NY, USA: ACM, 2013, pp. 2:1–2:6.

[23] R. S. Xin, D. Crankshaw, A. Dave, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: Unifying data-parallel and graph-parallel analytics," *CoRR*, vol. abs/1402.2394, 2014. [Online]. Available: http://arxiv.org/abs/1402.2394

[24] L. A. Adamic and B. A. Huberman, "Zipf's law and the internet," *Glottometrics*, vol. 3, pp. 143–150, 2002.

[25] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, "Graph structure in the web," *Comput. Netw.*, vol. 33, no. 1-6, pp. 309–320, Jun. 2000.

[26] S. Lattanzi and Y. Singer, "The power of random neighbors in social networks," in *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, ser. WSDM '15. New York, NY, USA: ACM, 2015, pp. 77–86.

[27] G. H. Golub and C. F. van Loan, *Matrix Computations*, 4th ed. JHU Press, 2013.

[28] B. Nadler, S. Lafon, R. R. Coifman, and I. G. Kevrekidis, "Diffusion maps, spectral clustering and eigenfunctions of fokker-planck operators," in *in Advances in Neural Information Processing Systems 18*. MIT Press, 2005, pp. 955–962.

[29] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[30] E. Amigó, J. Gonzalo, J. Artiles, and F. Verdejo, "A comparison of extrinsic clustering evaluation metrics based on formal constraints," *Inf. Retr.*, vol. 12, no. 4, pp. 461–486, Aug. 2009.

[31] D. Lewis, "Reuters-21578 text categorization test collection distribution 1.0," Available at http://archive.ics.uci.edu/ml/machine-learning-databases/reuters21578-mld/reuters21578.html, 26 September 1997.

[32] C. van Rijsbergen, S. Robertson, and M. Porter, "New models in probabilistic information retrieval," 1980.

[33] G. W. Snedecor and W. G. Cochran, *Statistical Methods*, 8th ed. Iowa State University Press, 1989.